ON THE SYNTHESIS OF FINITE-STATE ACCEPTORS

by

A. W. Biermann and J. A. Feldman

Computer Science Department

Stanford University

ABSTRACT: Two algorithms are presented for solving the following
            problem: Given a finite-set S of strings of symbols,
            find a finite-state machine which will accept the strings
            of S and possibly some additional strings which
            "resemble" those of S . The approach used is to
            directly construct the states and transitions of the
            acceptor machine from the string information. The
            algorithms include a parameter which enable one to
            increase the exactness of the resulting machine's
            behavior as much as desired by increasing the number of
            states in the machine. The properties of the algorithms
            are presented and illustrated with a number of examples.

            The paper gives a method for identifying a finite-state
            language from a randomly chosen finite subset of the
            language if the subset is large enough and if a bound
            is known on the number of states required to recognize
            the language. Finally, we discuss some of the uses of the
            algorithms and their relationship to the problem of
            grammatical inference.

Table of Contents

ON THE SYNTHESIS OF FINITE-STATE ACCEPTORS

by

A. W. Biermann and J. A. Feldman

1.   Introduction

An acceptor is a finite-state machine which receives strings of
symbols as input and which responds to each string with an answer of
either "yes" or "no"; that is, it accepts or rejects each string. This
paper discusses the problem of constructing an acceptor for a particular
finite set S of strings and perhaps some additional strings which
"resemble" those in S . We present two algorithms for constructing
such a machine from S and from additional information about the
required preciseness of the machine's behavior. The algorithms
presented enable one to obtain varying degrees of accuracy with
corresponding varying degrees of machine complexity. Thus, if the
acceptor is required to accept only the strings of S and no others,
it can be expected to require many more states than if a large number of
"extra" strings are allowed to be in the accepted set.

There are a number of finite-state machine synthesis algorithms in
the literature. Huffman [9], Mealy [10], and others have developed
algorithms for sequential machine design when some kind of transition
table or state diagram is given. Ott and Feinstein [11], Brzozowski [1],
and others have given methods for constructing acceptors from their

regular expressions. This paper is concerned with the problem of designing a finite-state acceptor when no simple transition table, state diagram, or regular expression is available.

Ginsburg [5, 6] gives an algorithm for synthesizing sequential machines from input-output behavior, a problem similar to the one dealt with here. However, our algorithms are concerned with the design of a different type of device, an acceptor, and the methods presented are distinctly different from those of [5, 6].

The techniques which are described here grew out of an idea by Feldman [2] who was attempting to infer finite-state grammars for sets of strings. Feldman's idea suggested a method of creating states and transitions from string information, and this concept became the core of the algorithms which were subsequently developed.

In this paper we will show how to construct a machine $A(S,k)$ which is an acceptor of set S (Section 2). Other properties of $A(S,k)$ will be investigated with examples given (Section 3), and its applications will be discussed (Section 4). Finally, a second algorithm and its properties will be investigated (Section 5).

2. A Finite-State Acceptor

We introduce a number of definitions largely following the notation of Ginsburg [6].

Definition 2.1. A nondeterministic automaton A is a five-tuple $\langle Q, \Sigma, f, Q_o, F \rangle$ where

Q is a finite nonempty set (of states)

Σ is a finite nonempty set (of input symbols)

f is a mapping $Q \times \Sigma \to 2^Q$ (the transition function)

Q_o is a subset of Q (the set of initial states)

F is a subset of Q (the set of final states) .

The function f is extended to a mapping $Q \times \Sigma^* \to 2^Q$ by the recursive definition

$$f(q,\Lambda) = \{q\}$$

where Λ is the string of length zero and $q \epsilon Q$, and

$$f(q,wa) = \bigcup_{q' \epsilon f(q,w)} f(q',a)$$

where $w \epsilon \Sigma^*$ and $a \epsilon \Sigma$.

Definition 2.2. The <u>language L(A)</u> of the nondeterministic automaton A will be defined to be the set of strings $w = a_1 a_2 \cdots a_j$, $a_i \epsilon \Sigma$ for $1 \leq i \leq j$, such that there is a sequence of states $q_o, q_1, \ldots, q_j$ with the properties

(1) $q_o \epsilon Q_o$

(2) $q_i \epsilon f(q_{i-1}, a_i)$ for $1 \leq i \leq j$

(3) $q_j \epsilon F$.

We will be interested in the relation of the languages of various automata to a fixed set S of strings. If $S \subseteq L(A)$, then A will be said to <u>accept</u> S . If $S = L(A)$, then A will be said to <u>accept exactly</u> S .

After a preliminary definition, we will show how to construct a class of automata which will be shown to accept S .

Definition 2.3. The k-tail of z with respect to $S \subseteq \Sigma^*$ will be denoted as $g(z,S,k)$ and will be defined as follows: Let $z \epsilon \Sigma^*$ be such that $zw \epsilon S$ for some $w \epsilon \Sigma^*$, and let k be a nonnegative integer. Then $g(z,S,k)$ is defined as the set of strings $w \epsilon \Sigma^*$ with the properties

(a) $zw \epsilon S$

(b) $length(w) \leq k$.

$g(z,S,k)$ is undefined if z and k are outside of the domains specified.

The acceptor of set S will be determined from the set S and from the look-ahead level k and will be denoted $A(S,k)$.

Definition 2.4. If S is a finite set of strings from Σ^* , let $A(S,k)$ be the nondeterministic automaton

$$A(S,k) = \langle Q, \Sigma, f, Q_0, F \rangle$$

where

$$Q = \{ q \epsilon 2^{\Sigma^*} \mid g(z,S,k) = q \text{ for some } z \epsilon \Sigma^* \}$$

Σ = a finite nonempty set of input symbols

$$f(q,a) = \{ q' \epsilon Q \mid \text{ there is a } z \epsilon \Sigma^* \text{ such that } g(z,S,k) = q$$
$$\text{and } g(za,S,k) = q' \}$$

$$Q_0 = \{ g(\Lambda,S,k) \}$$

$$F = \{ q \epsilon Q \mid \Lambda \epsilon q \} \quad .$$

4

The machine $A(S,k)$ thus has as states the set of all k-tails which can be constructed from S . A transition from k-tail S_1 to k-tail S_2 under input symbol b will occur if there is a string z with k-tail S_1 and zb has k-tail S_2 .

The set S will be said to <u>yield</u> the language $L(A(S,k))$ (at look-ahead level k) .

<u>Example 2.1.</u> Suppose as an illustration of Definition 2.4 that we consider the example where $S = \{a,ab,abb\}$ and $k = 1$. Then $A(S,k) = \langle Q,\Sigma,f,Q_0,F\rangle$ where

$$Q = \{\{a\},\{\Lambda,b\},\{\Lambda\}\}$$

$$\Sigma = \{a,b\}$$

$$f = (\{a\},a) = \{\{\Lambda,b\}\}$$

$$f = (\{\Lambda,b\},b) = \{\{\Lambda\},\{\Lambda,b\}\}$$

$$Q_0 = \{\{a\}\}$$

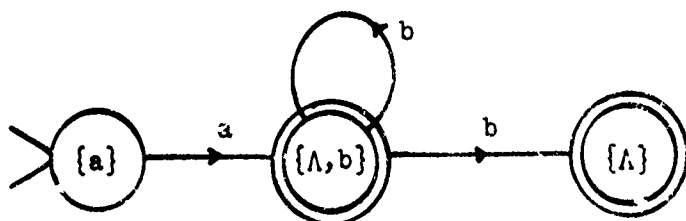$$F = \{\{\Lambda\},\{\Lambda,b\}\}$$

FIGURE 1   Example 2.1

key

initial state:

final state:

5

Note that the resulting machine which is diagramed in Figure 1 recognizes the set ab^*. $A(S,k)$ is typically not in minimal form as is the case here, but minimization can always be done by well known algorithms [4, 5, 6, 8]. If k had been set at 2 or larger, we would obtain $L(A(S,k)) = S$.

Theorem 2.1. $S \subseteq L(A(S,k))$ for all nonnegative k.

Proof. If $w = a_1a_2a_3 \ldots a_j \in S$, $a_i \in \Sigma$ then let $q_0 = g(\Lambda,S,k)$ and $q_i = g(a_1a_2 \ldots a_i,S,k)$ for $i = 1,2,3,\ldots,j$. The sequence of states $q_0,q_1,q_2,\ldots,q_j$ satisfy the three properties of Definition 2.2 so we have $w \in L(A(S,k))$. This completes the proof.

The construction of Definition 2.4 has provided states to account for all possible k-continuations of heads of strings in S so $A(S,k)$ will surely be able to accept S. The next section is concerned with stronger requirements on $A(S,k)$ and the consequences of varying the look-ahead level k.

3. Further Properties of $A(S,k)$

The machine $A(S,k)$ will have no more than $2^{\sum_{i=0}^{k} m^i}$ states if S is from an alphabet Σ of m distinct symbols so that the upper bound on a machine's size can be adjusted by setting the value of k. Thus we can expect $A(S,k)$ to increase greatly in "computing power" as k is made larger. For example, if $L(A(S,k))$ is considered to be an

approximation to S , we can expect the approximation to be much better if k is larger and, in fact, $S = L(A(S,k))$ if k is as large as the length of the longest string in S as will be shown below. From another point of view, we can consider $L(A(S,k))$ to be a guess of the language L_0 from which the sample S has been chosen. If k is very small, the "guess" of L_0 will constitute a very liberal inference and may include most of the strings from the alphabet of S . If k is as large as the longest string in S , however, the inference will be very conservative and will, in fact, include only the strings of S . All of this will be made precise in the paragraphs that follow.

The first property to confirm is that $L(A(S,k)) = S$ if k is large enough, and the proof will make use of the following obvious Lemma.

Lemma 3.1. Let $h(z,S) = \{w\epsilon\Sigma^* \mid zw\epsilon S\}$. Then if k is greater than or equal to the length of the longest string in S , $g(z,S,k) = h(z,S)$ for all $z\epsilon\Sigma^*$ such that $g(z,S,k)$ is defined.

Theorem 3.1. $L(A(S,k)) = S$ if k is greater than or equal to the length of the longest string in S .

Proof. Employing the Lemma and the definition of $A(S,k)$, we have $A(S,k) = \langle Q = \{q\epsilon 2^{\Sigma^*} \mid q = h(z,S)\}, \Sigma, f, Q_0 = \{h(\Lambda,S)\}, F = \{q\epsilon Q \mid \Lambda\epsilon h(z,S)\}\rangle$ where $f(q,a) = \{q'\epsilon Q \mid$ there is $z\epsilon\Sigma^*$ such that $q = h(z,S)$ and $q' = h(za,S)\}$ or $f(h(z,S),a) = \{h(za,S)\}$ if $h(za,S)$ is not empty. It follows that

$f(h(z,S),w) = \{h(zw,S)\}$ for all $w\epsilon\Sigma^*$ if $h(zw,S)$ is not empty. Then $f(h(\Lambda,S),w) = \{h(w,S)\}$. But $h(\Lambda,S)$ is the initial state so w will be accepted by $A(S,k)$ if and only if $h(w,S)$ is a final state. That is true if and only if $\Lambda\epsilon h(w,S)$ which holds if and only if $w\epsilon S$. We conclude that $w\epsilon L(A(S,k))$ if and only if $w\epsilon S$ which completes the proof.

The lower bound given on k is in general the best bound which can be obtained as can be seen by applying the construction of Definition 2.4 to the set $S = \{\Lambda,a,a^2,\ldots,a^i\}$. It may also be worth mentioning that the machine of Theorem 3.1 will be deterministic and in its minimal form.

We will next investigate the languages $L(A(S,k))$ as k is varied and note their relationship to each other. It turns out that $L(A(S,k))$ "covers" $L(A(S,k+i))$ in the sense of Reynolds [12] for nonnegative i. In fact, the next theorem could be derived from Reynold's results.

Theorem 3.2. $L(A(S,k+1)) \subseteq L(A(S,k))$.

Proof. Assume $w = a_1 a_2 \ldots a_j \epsilon L(A(S,k+1))$, $a_i\epsilon\Sigma$ for $1 \leq i \leq j$. Then there is a sequence of states $q_0,q_1,\ldots,q_j$ in $A(S,k+1)$ with the properties

(1) $\{q_0\} = Q_0$

(2) $q_{i+1} \epsilon f(q_i,a_{i+1})$ for $0 \leq i \leq j-1$

and

8

(3) $\quad q_j \epsilon F \quad$ (i.e., $\Lambda \epsilon q_j$) .

Furthermore, there is a string $z_i \epsilon \Sigma^*$ such that $z_i w_i \epsilon S$ for some $w_i \epsilon \Sigma^*$ where $q_i = g(z_i, S, k+1)$ and $q_{i+1} = g(z_i a_{i+1}, S, k+1)$ for each $i = 0, 1, 2, \ldots, j-1$. Next consider $A(S, k)$ and the sequence of states $q_0', q_1', \ldots, q_j'$ where $q_i' = q_i - \{$all strings in q_i of length $k+1\}$. These states certainly exist in $A(S, k)$ and have properties analogous to (1), (2), and (3) above. To justify (2), for example, simply employ the z_i's defined above and observe that $q_i' = g(z_i, S, k)$ and $q_{i+1}' = g(z_i a, S, k)$ will occur so that $q_{i+1}' \epsilon f(q_i', a_{i+1})$ for $0 \le i \le j-1$. Therefore $w \epsilon L(A(S, k))$ and the proof is complete.

The last three theorems combine to give a good picture of how S is related to $L(A(S, k))$ as k is varied, and Figure 2 illustrates the situation. Each language $L(A(S, k))$ will include S and will be included by $L(A(S, k-1))$. k is thus a parameter of the algorithm which can be used to adjust $L(A(S, k))$ to be as close to S as desired at the cost of increasing the number of states in the acceptor.

9

There exists some $m \leq$ length of longest string in S

such that $L(A(S, \ell)) = S$ if $\ell \geq m$.
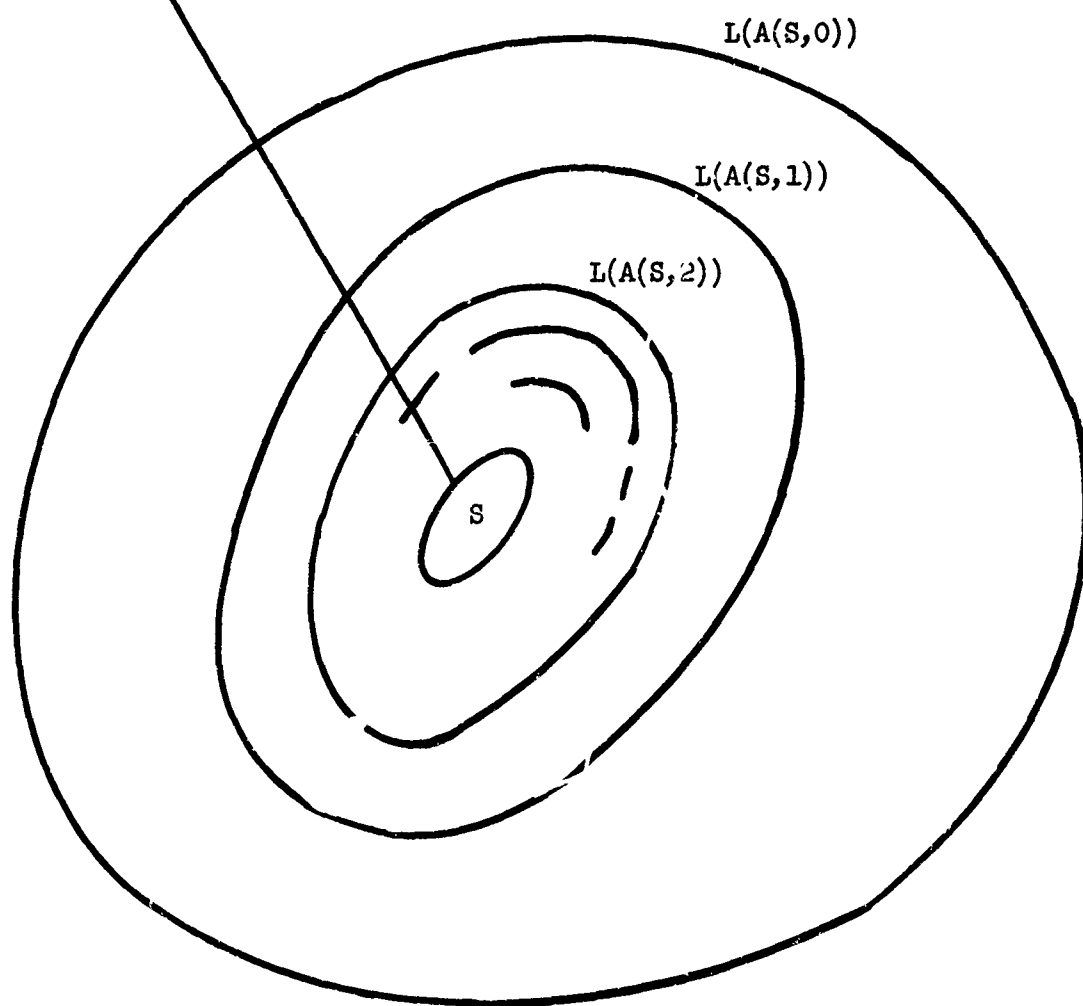
$L(A(S,0))$

$L(A(S,1))$

$L(A(S,2))$

S

Figure 2. The relationships between the languages $L(A(S,k))$

as k is varied.

Suppose that the set of strings S is chosen from some finite-state language L_0. It is desired to know what relationship the language $L(A(S,k))$ may have to L_0 and under what conditions one can expect, for example, equality between the languages. Some information can be obtained by turning the problem around and asking the following question: If we are given finite-state language L_0, how do we construct a finite set S and how do we set k in order to obtain $L(A(S,k)) = L_0$? The answer is to consider the minimal deterministic automaton M which will accept exactly L_0 and to construct S in such a way that $A(S,k)$ will be equivalent to M. If M has n states then we set the look-ahead level k to equal $n-2$ since the states of M can be characterized by their behavior $n-2$ steps into the future. This analysis will now be formally carried out.

Definition 3.1. A finite-state deterministic automaton M is a five-tuple $\langle P, \Sigma, d, p_0, D \rangle$ where

$\quad$ P $\quad$ is a finite nonempty set (of states)

$\quad$ Σ $\quad$ is a finite nonempty set (of input symbols)

$\quad$ d $\quad$ is a mapping $P \times \Sigma \to P$ (the transition function)

$\quad$ $p_0 \epsilon P$ $\quad$ (the initial state)

$\quad$ D $\quad$ is a subset of P (the set of final states).

The function d is extended to a mapping $P \times \Sigma^* \to P$ as the function f was above.

11

Definition 3.2. The language $L(M)$ of the deterministic
automaton M will be defined as

$$L(M) = \{w\epsilon\Sigma^* \mid d(p_0,w)\epsilon D\} \quad .$$

Definition 3.3.

(1) A state p_i in a machine M will be called l-reachable
if there is a string w of length l or less such that $d(p_0,w) = p_i$.

(2) The states of machine M will be called k-distinguishable
if for each pair of distinct states p_i and p_j in M there is a
string w with length $(w) \leq k$ such that exactly one of the states
$d(p_i,w)$ or $d(p_j,w)$ is in D .

(3) Consider the set $\mathcal{C}$ of machines M whose states are
all l-reachable and k-distinguishable. Then $Z(l,k)$ will be defined
as follows:

$$Z(l,k) = \{L_0 \mid L_0 = L(M) \quad \text{and} \quad M\epsilon\mathcal{C}\} \quad .$$

Definition 3.4. The symbol M will henceforth be used to
designate the minimal deterministic machine such that $L(M) = L_0$
where L_0 is the language we are considering at the moment.

Definition 3.5. If $w_i\epsilon\Sigma^*$ then $w_0 \cdot \{w_i \mid i = 1,2,\ldots,j\} =$
$\{w_0w_i \mid i = 1,2,3,\ldots,j\}$.

Theorem 3.3. If $L_0 \epsilon Z(l,k)$ then $L_0 = L(A(S,k))$ if S is
constructed as follows:

12

(1) Choose strings $z_1, z_2, \ldots, z_m$ such that for every state p

in M (where $L(M) = L_0$) there is a $z_i = u_i v_i$ such that $d(p_0, u_i) = p$.

(2) $S = \displaystyle\bigcup \quad u \cdot g(u, L_0, k+1+\sigma_u)$

$\qquad u \epsilon \Sigma^*$ such that

$\qquad z_i = uv$ for some z_i

where

$\qquad \sigma_u = 1$ if $g(ua, L_0, k) = \varphi$ and $g(ua, L_0, k+1) \neq \varphi$ for $a \epsilon \Sigma$

$\qquad \sigma_u = 0$ otherwise.

φ designates the empty set.

Proof.   An informal justification will be included here, and a
more detailed proof will appear in the Appendix.

S is constructed so that each state p in M has a counterpart
in $A(S,k)$, namely the k-tail $g(u, L_0, k)$ where $d(p_0, u) = p$.
Furthermore, each successor to p , specifically $p' = d(p,a)$ $(a \epsilon \Sigma)$
must have a counterpart $g(ua, L_0, k)$ in $A(S,k)$ which is a successor
(under a) to $g(u, L_0, k)$. To guarantee that $A(S,k)$ will contain
$g(u, L_0, k)$ and all of its successors, the set $u \cdot g(u, L_0, k+1)$ is
included in S . The set of u's is defined so that every state p in
M will have a counterpart in $A(S,k)$ along with correctly assigned
successors. (For the moment, we ignore the quantity σ_u .)

The fact that $A(S,k)$ will accept L_0 is clear because its
construction has insured its ability to simulate M . However, many

13

other states and transitions may appear in $A(S,k)$ so one might think $A(S,k)$ would accept strings which are not in L_0. This will not occur because the only other states which will appear in $A(S,k)$ will have the form $g(u,L_0,k-i)$ with $0 \leq i \leq k$. The transitions will be such that $A(S,k)$ will be in state $g(u,L_0,k-i)$ only when $A(S,k)$ is also in state $g(u,L_0,k)$. (Remember that $A(S,k)$ is nondeterministic.) Therefore no strings will be accepted by $A(S,k)$ which are not also accepted by M.

The term σ_u arises in the case where one of the successors $g(ua,L_0,k)$ to a state $g(u,L_0,k)$ is the empty set φ. Then $A(S,k)$ may not include the transition $f(g(u,L_0,k),a) = g(ua,L_0,k)$ because $g(ua,L_0,k)$ may not have been created as a successor to $g(u,L_0,k)$. This problem is remedied by including $u \cdot g(u,L_0,k+1+\sigma_u)$ in S where $\sigma_u = 1$. The reader is referred to the appendix for a detailed proof of the theorem.

Definition 3.6. A set $S \subseteq L_0$ which is constructed as described by Theorem 3.3 will be called **special** (_for language_ L_0 _at level_ _k_).

Example 3.1. As an example, consider the automaton of Figure 3 which accepts all of the strings on a three letter alphabet which have exactly one A .
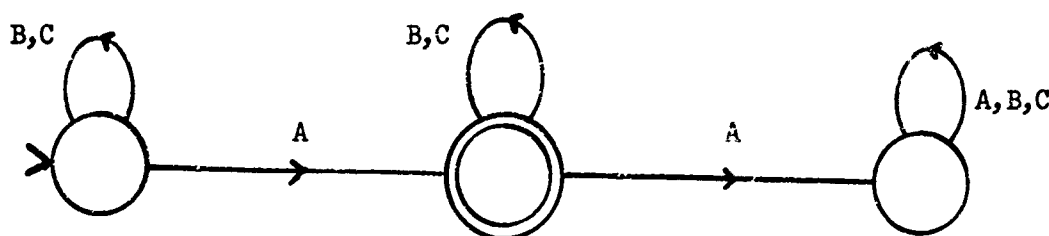
14

Figure 3. Example 3.1.

Only one string $z_1 = AA$ is necessary to satisfy (1) of Theorem 3.3.
Letting $u = \Lambda$, $u' = A$, $u'' = AA$, and $k = 1$, we obtain

$$S = u \cdot g(u, L_0, k+1) \cup u' \cdot g(u', L_0, k+1) \cup u'' \cdot g(u'', L_0, k+1)$$

$= \{A, AB, AC, BA, CA\} \cup \{ABB, ABC, ACB, ACC\} \cup \{ \ \}$. This results in the
machine $A(S,1)$ which accepts exactly the desired language.

Theorem 3.3 is important because it indicates about how much
information must be obtained from L_0 before it can be recognized by
the algorithm. For most problems, Theorem 3.3 will, with appropriately
chosen z_i , give the S with the smallest possible number of strings
such that S will yield L_0 . It is true that there are sets S
which yield L_0 and which are not special, but they are generally
larger. A more precise characterization of the sets which yield L_0
is possible, but it is too complex to be included here.

It will also be noted but not proved that if S is special for L_0
and $S \subseteq S' \subseteq L_0$, then $L_0 = L(A(S,k)) \subseteq L(A(S',k))$. The inclusion
of a special set in S' insures that states and transitions will appear

15

in $L(A(S',k))$ which will accept all of the strings in L_j . The additional strings in S' may add states and transitions to $L(A(S',k))$ which cause it to accept strings which are not in L_0 . Thus a randomly chosen subset of L_0 which contains a special set will yield a language which contains L_0 .

Two useful corollaries follow from Theorem 3.3.

Definition 3.7. $S|_i = \{w \in S \mid \text{length } (w) \leq i\}$.

Corollary 3.1.   If $L_0 \in Z(\ell,k)$ then $L_0 = L(A(L_0|_i,k))$ if $i \geq \ell+k+1+\sigma$ where $\sigma = 1$ if the state $q = \{ \ \}$ is in $A(L_0|_{i-1},k)$ and $\sigma = 0$ otherwise.

Proof.   Choose the strings $z_1, z_2, \ldots, z_m$ in (1) of Theorem 3.3 to be all the strings in L_0 which have length $i-k-1-\sigma$. Then S as defined in (2) will be exactly $L_0|_i$.

Corollary 3.2.   If $L_0 = L(M)$ where M is an n-state automaton, then $L_0 = L(A(L_0|_i,n-2))$ if $i \geq 2n-2+\sigma$ where $\sigma = 1$ if the state $q = \{ \ \}$ is in $A(L_0|_{i-1},n-2)$ and $\sigma = 0$ otherwise.

Proof.   Note that the language for any n-state automaton is in $Z(n-1,n-2)$ and employ the previous corollary.

The lower bound on i in Corollary 3.2 if $o = 0$ is the best
performance that could be hoped for under any conditions. That is,
there exist distinct n-state languages which are identical for all
strings of length $2n-3$ or less so that no algorithm could be expected
to discover L_0 with only that much information. The two n-state
machines of Figure 4 provide an example.
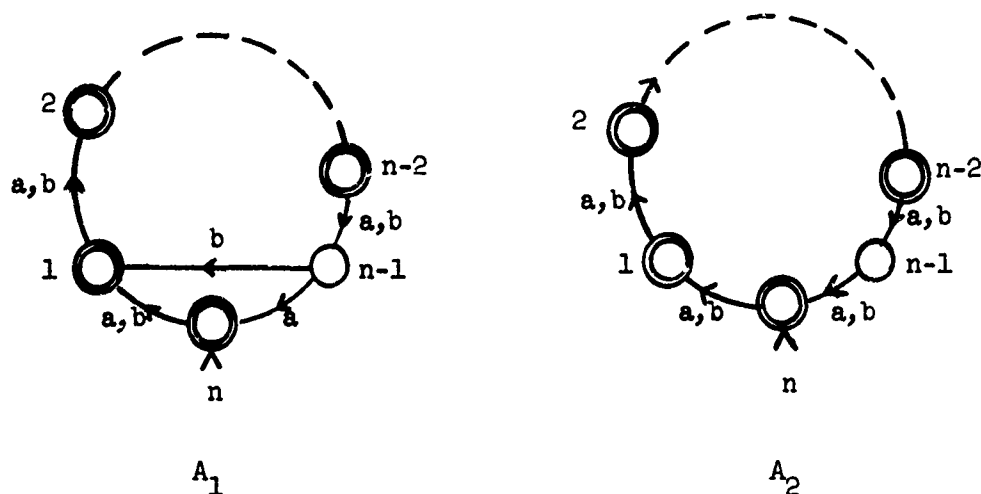
Figure 4. $L(A_1)\big|_{2n-3} = L(A_2)\big|_{2n-3}$ but $L(A_1) \neq L(A_2)$.

Example 3.2. This section will be concluded with an example which
illustrates the various results given above. All of the strings of length
five or less for a particular finite state language L_0 in $Z(2,2)$ are
input to the algorithm while look-ahead level is varied. The set of
strings S satisfies the requirements of Theorem 3.3 so that $L_0 = L(A(S,2))$.
The resulting machines are shown in Figure 5. The reader may check that
the various assertions made above do indeed hold.

17

k	A(S,k)	L(A(S,k))

$k = 0$ ... $(A+B)^*B$

$k = 1$ ... $[A(A+B)^*B+B(A+B)(A+B)^*B]^*B$

$k = 2$ ... $(AB^*AA^*B+B(A+B)B^*AA^*B)^*B$

$k = 3$ ... $((A+BA+BB)B^*(AB+AAB))^*B$

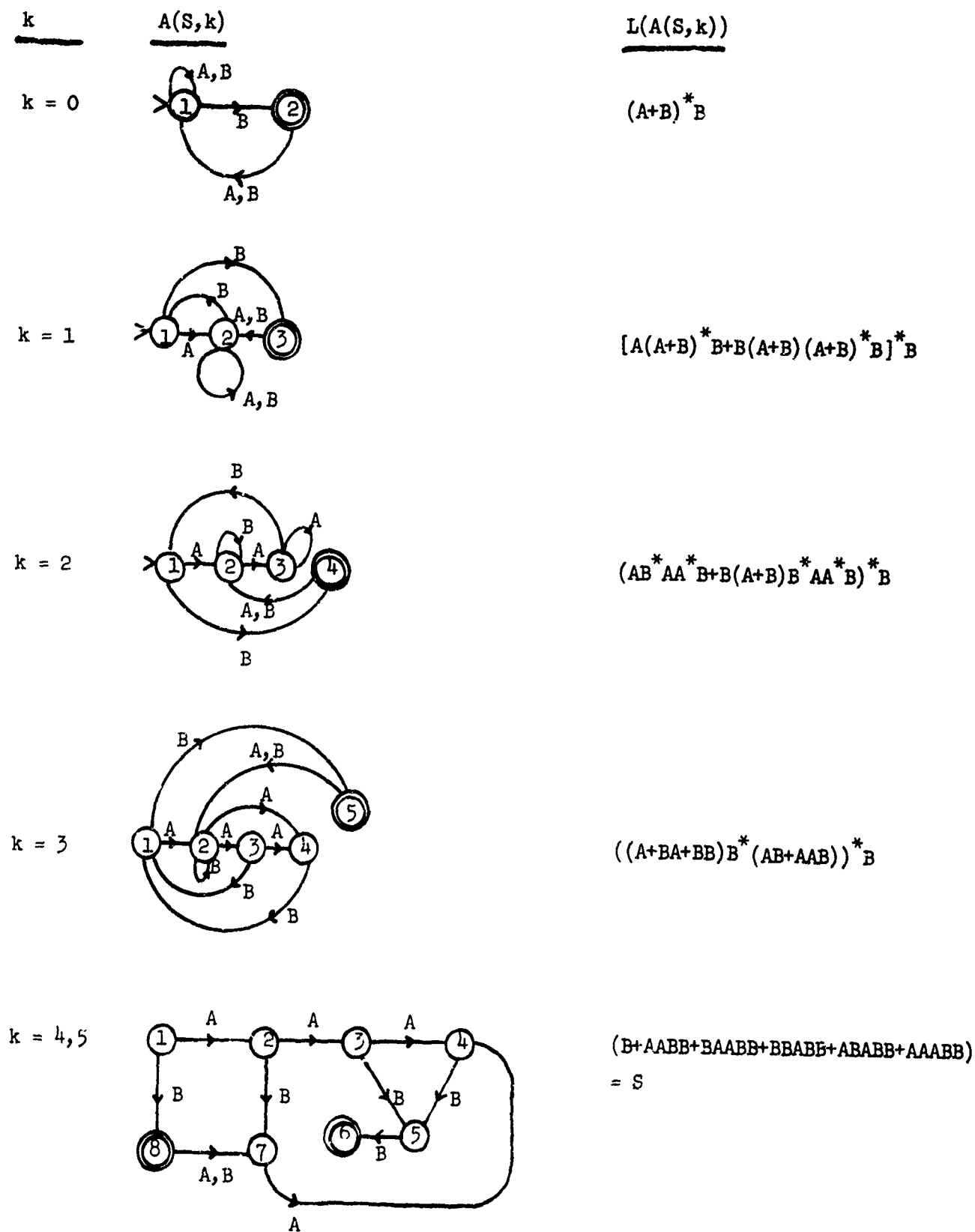$k = 4,5$ ... $(B+AABB+BAABB+BBABB+ABABB+AAABB)$ $= S$

Figure 5. Example 3.?. $S = \{B, AABB, BAABB, BBABB, ABABB, AAABB\}$

18

4. Applications

The results of the previous sections are applicable to a variety
of problems in the synthesis of automata. The non-deterministic
automata created by these methods can, of course, be converted to
minimal deterministic automata by standard techniques [4, 5, 6, 8].
We will sketch briefly algorithms for some synthesis problems.

Suppose one is given two disjoint finite sets S_1 and S_2 and
asked to construct a machine which accepts S_1 and not S_2 . The
required machine is the one of minimum k where $y \epsilon S_2$ implies
$y \notin L(A(S_1,k))$. This construction also solves the problem of
constructing a machine given both strings and non-strings of its
language.

If one is given a bound n on the number of states of a finite
state machine and all of the strings of its language L_0 are available,
the minimal machine for L_0 can be found. One uses $k = n-2$ and
$S = L_0|_{2n-1}$ and computes $A(S,k)$ as described above. By Corollary 3.2,
$L(A(S,k)) = L_0$; the minimized version of $A(S,k)$ is the minimal machine
for the language L_0 .

One can also solve the problem of finding the minimum machine for
distinguishing the disjoint infinite finite-state languages L_1 and L_2
when given bounds n_1, n_2 on the sizes of their respective machines.
The construction of the paragraph above gives minimal machines M_1, M_2
for L_1 and L_2 respectively. The smaller of these machines is an
upper bound on the size of the desired machine.

19

The most important use of our algorithms occurs in the sequential learning situation. Suppose one is given a new string $y_i \epsilon L_0$ at time t_i and asked to select a machine A_i which describes the sequence up to time t_i. This is the analog of the grammatical inference problem [2, 3] which was our original motivation and is a model of scientific reasoning and other hypothesis forming behaviour. The interesting questions in sequential learning are the nature of the machines A_i and the limiting behaviour of an algorithm as $i \to \infty$. We have developed elsewhere [3] a number of general results on this subject. These show that there is an algorithm which will choose the best A_i at each i and will be such that the successive A_i become ever better approximations to the machine of L_0.

There are two important advantages of the methods of this paper over the general algorithms described in [3]. The latter methods depend on enumerating all finite-state machines in order, while the construction of Section 2 requires one to consider only a small number of machines. In addition, for fixed k, the machine $A(S_i,k)$ is easily constructed from the machine $A(S_{i-1},k)$. This notion of sequential modification of a synthesized machine is extremely important and will be briefly described.

Suppose that $A(S_{i-1},k)$ has been constructed and that $A(S_i,k)$ must now be found; $S_i = S_{i-1} \cup \{y_i\}$. Let $y_i = a_1 a_2 a_3 \cdots a_r$, $a_j \epsilon \Sigma$. Then only the $k+1$ (or fewer) states $g(a_1 a_2 a_3 \cdots a_{r-k}, S_{i-1}, k)$, $g(a_1 a_2 a_3 \cdots a_{r-k+1}, S_{i-1}, k)$, $\ldots$, $g(a_1 a_2 \cdots a_r, S_{i-1}, k)$ in $A(S_{i-1},k)$ are affected by the addition of the string y_i. Each of the sets $g(a_1 a_2 a_3 \cdots a_{r-k+j}, S_{i-1}, k)$ must have the string $a_{r-k+j+1} a_{r-k+j+2} \cdots a_r$

added to it, and the transition function f must be correspondingly
changed. These alterations in $A(S_{i-1},k)$ are enough to produce the
new machine $A(S_i,k)$. Considering the Example 2.1, if the string abba
is added to S, the new acceptor can be constructed by altering the
states $g(abb,S,1)$ and $g(abba,S,1)$. $g(abb,S,1) = \{\Lambda\}$ becomes $\{\Lambda,a\}$,
$g(abba,S,1)$ which was not defined in the original construction becomes $\{\Lambda\}$,
and $A(S \cup \{abba\},1)$ now accepts exactly the set $ab^* + ab^*ba$.

We have shown that the construction of Definition 2.4 will produce
machines $A(S_i,k)$ which have desirable properties. It remains to
describe an algorithm for choosing k and deciding which $A(S_i,k)$ to
call the machine A_i. Suppose one is given an upper bound n on the
number of states required of M where $L_0 = L(M)$. Consider the
following algorithm:

<u>Algorithm 4.1.</u>   At each i, compute $B_i = A(S_i|_{2n-1}, n-2)$. If
$S_i \subseteq L(B_i)$ then $A_i = B_i$, otherwise $A_i = A(S_i, n-2)$.

<u>Theorem 4.1.</u>   Algorithm 4.1 has the following properties:

    a) $S_i \subseteq L(A_i)$.

    b)   If S_i contains a special set, $L_0 \subset L(A_i)$.

    c)   If $L_0|_{2n-1} \subseteq S_j$ for some j then for all $i \geq j$,
         $L_0 = L(A_i)$.

Proof. Part a) follows directly from Theorem 2.1, part b) from the discussion following Theorem 3.3, part c) from Corollary 3.2.

If one assumes, as seems reasonable, that every string in L_0 will occur as some y_i , Algorithm 4.1 will eventually choose only a machine which generates exactly L_0 . This is known in the literature as the algorithm identifying L_0 . There is a problem in that Algorithm 4.1 depended on an a priori estimate of n , the size of a machine for L_0 . It is shown in [3] that without this estimate, no algorithm will be able to identify the finite-state languages from an arbitrary presentation of L_0 . If, however, the presentation includes the information about which strings are not in L_0 or meets certain regularity conditions, there are algorithms like 4.1 which will identify the finite-state languages.

Although there is a close relation between the problem discussed here and the grammatical inference problem [2, 3], the criteria for the best solution to the problem are quite different and this leads to a number of other differences in the two studies.

5. Another Finite-State Acceptor

Another machine $B(S,k)$ which accepts S at look-ahead level k will be introduced here and discussed. The properties of $B(S,k)$ are not as easily characterized or as nice as those of $A(S,k)$, but it does handle certain problems better than $A(S,k)$ and so will be briefly discussed here.

<u>Definition 5.1.</u> Assume that $z\epsilon\Sigma^*$, $zy\epsilon S$ for some $y\epsilon\Sigma^*$, $S \subseteq \Sigma^*$, and k is a nonnegative integer. $m(z,S,k)$ will be defined as the set of strings $w\epsilon\Sigma^*$ with the properties

    (1) $zwx \in S$ for some $x\epsilon\Sigma^* - \{\Lambda\}$

    (2) length $(w) = k$.

<u>Definition 5.2.</u> $e(z,S,k) = m(z,S,k) \cup g(z,S,k) \cdot \#$ where $\{w_1,w_2,\ldots,w_i\} \cdot y$ is defined to be $\{w_1y,w_2y,\ldots,w_iy\}$. $e(z,S,k)$ is undefined if z and k are outside of the domains specified in the definition of $g(z,S,k)$.

<u>Definition 5.3.</u> If S is a finite set of strings from Σ^* , let $B(S,k)$ be the finite nondeterministic automaton

$$B(S,k) = \langle Q,\Sigma,f,Q_0,F\rangle$$

where

$Q = \{q\epsilon2^{\Sigma^* \cdot \#} \cup 2^{\Sigma^*} |$ there is a z with $zw\epsilon S$ and $q = e(z,S,k)\}$

$\Sigma =$ a finite nonempty set of input symbols

$f(q,a) = \{q'\epsilon Q |$ there is a $z\epsilon\Sigma^*$ such that

$$e(z,S,k) = q \text{ and } e(za,S,k) = q'\}$$

$Q_0 = \{e(\Lambda,S,k)\}$

$F = \{q\epsilon Q | \#\epsilon q\}$.

The proofs of the following theorems are nearly identical to their respective counterparts of Sections 2 and 3 and will not be repeated here.

<u>Theorem 5.1.</u> $S \subseteq L(B(S,k))$ for all nonnegative k .

<u>Theorem 5.2.</u> $L(B(S,k)) = S$ if k is greater than or equal

to the length of the longest string in S .

<u>Theorem 5.3.</u> $L(B(S,k+1)) \subseteq L(B(S,k))$.

There is no immediate analogy to Theorem 3.3 for $B(S,k)$.

Since the states of $B(S,k)$ contain more information than those

of $A(S,k)$, it is not surprising to find the following true:

<u>Theorem 5.4.</u> $L(B(S,k)) \subseteq L(A(S,k))$.

<u>Proof.</u> $w = a_1 a_2 \ldots a_j \in L(B(S,k))$, $a_i \in \Sigma$, implies the existence

of states $q_0, q_1, q_2, \ldots, q_j$ in $B(S,k)$ with the properties (1), (2),

and (3) of Definition 2.2. This implies the existence of states

$q_0', q_1', q_2', \ldots, q_j'$ in $A(S,k)$ which also satisfy (1), (2), and (3).

Let $q_i' = \{w | w\# \in q_i\}$ for $i = 0, 1, 2, \ldots, j$. Therefore $q_{i+1} \in f(q_i, a_{i+1})$

in $B(S,k)$ implies there is a $z \in \Sigma^*$ such that $e(z,S,k) = q_i$ and

$e(za_{i+1}, S, k) = q_{i+1}$. From the definition of e , this implies that

$g(z,S,k)$ and $g(za_{i+1}, S, k)$ exist as states in $A(S,k)$, and we have

just named these states q_i' and q_{i+1}' , respectively. This leads to

$q_{i+1}' \in f(q_i', a_{i+1})$ in $A(S,k)$ which proves (2) of Definition 2.2 for

the states $q_0', q_1', \ldots, q_j'$.   (1) and (3) are easy to check and so it

follows that $w \in L(A(S,k))$.

Thus $B(S,k)$ accepts S and it accepts fewer "extra" strings
than $A(S,k)$. This effect is extreme in certain problems and may be
considered quite an advantage. However, it has not been shown that
there is any subset S of a language which will yield the language
using $B(S,k)$ as was shown for $A(S,k)$, so the earlier machine may
be preferred.

6. Discussion and Summary

This paper gives two algorithms for constructing acceptors from
finite sets of strings. Both algorithms have been programmed on a
computer and extensively tested. Typical constructions of machines
with ten or twenty states take a few seconds or less to complete.
Many other versions of these algorithms are possible, and some were
investigated although they are not described here.

The authors are not aware of a comparable solution to this problem
elsewhere in the literature. The algorithm has a simple operation, and
is therefore easy to program and fast in execution. The parameter k
enables the user to obtain as exact a fit to the needed behavior as he
desires at the cost of increasing the complexity of the resulting acceptor.
The simplicity of the algorithm makes its operation easy to understand
and easy to characterize. Finally, the system has the distinct advantage
that if a large amount of computational effort is invested in finding an
acceptor for a set of strings, changes can be made in its behavior without
the necessity of starting the design procedure over again. If $A(S,k)$ is
created to accept S and then S is changed slightly, only the states and
transitions in $A(S,k)$ which correspond to the changes in S need to be
adjusted to obtain a new acceptor.

25

Appendix

A more detailed proof of Theorem 3.3 will be included here.

__Definition A1.__   If p_i is a state in M with the property that $d(p_i,w)\notin D$ for all $w\in\Sigma^*$, then p_i will be called an __absorbing state__.

__Lemma A1.__ $L(A(S,k)) = L_0$ if and only if for each state $p_i\in P$ of M there is a set $\{x_{i,1},x_{i,2},\dots,x_{i,j_i}\}$ of distinct sets $x_{i,j}$ of states $q\in Q$ of $A(S,k)$ such that

(1) $d(p_0,w) = p_i$ if and only if $f(q_0,w) = \{x_{i,1},\dots,x_{i,j_i}\}$
where $w\in\Sigma^*$, $\{q\} = Q$ unless p_i is an absorbing state.
$d(p_0,w) =$ absorbing state if and only if $f(q_0,w) = \varphi$.

(2) If $p_i\in D$ then $x_{i,j}\cap F\neq\varphi$ for $j = 1,2,\dots,j_i$.

(3) If $p_i\notin D$ then $x_{i,j}\cap F = \varphi$ for $j = 1,2,\dots,j_i$.

__Proof.__   Assume that the three conditions hold and observe why it follows that $L(A(S,k)) = L_0$. Construct the deterministic automaton $N = <X,\Sigma,e,x_1,E>$ which has the same behavior as $A(S,k)$. [6]

$X \subset 2Q$

$e(x,a) = x'$ if $x' = \bigcup_{q\in x} f(q,a)$

$x_0 = Q_0$

$E = \{x\,|\,x\cap F \neq \varphi\}$

The three conditions partition the states of N into sets Y_i of states which are equivalent to p_i in M. They require by (1) that N be in a state of Y_i if and only if M is in p_i and by (2) and (3) that N is in a final state if and only if M is in one. So M and N are equivalent by conditions (1), (2), and (3), and N and A are equivalent by construction. Therefore, $L(A) = L(M) = L_0$.

If $L(A(S,k)) = L_0$, then N can be defined as above and the sets $Y_i = \{x \epsilon X \mid x$ is equivalent to $p_i \epsilon P\}$ can be constructed. Certainly $d(p_0, w) = p_i$ if and only if $e(x_0, w) = x$ for one of the $x \epsilon Y_i$ since N must go into a state which is equivalent to p_i. So property (1) holds. $p_i \epsilon D$ implies $x \epsilon E$ which means $x \cap F \neq \varphi$. So property (2) holds and property (3) holds similarly.

Proof of Theorem 3.3. We construct $A(S,k)$ and show that it satisfies the conditions of Lemma A1. Corresponding to state $p_i \epsilon P$ of M we construct the sets $x_{i,j}$, $1 \leq j \leq j_i$, which each have the state $g(u_\ell, L_0, k)$ where $d(p_0, u_\ell) = p_i$ and $u_\ell v_\ell = z_\ell$ is one of the strings designated in (1). (If p_i is an absorbing state then construct only the set $x_{i,1} = \{\ \}$.) Each of the sets $x_{i,j}$, $1 \leq j \leq j_i$, also will contain either none or some of the states $g(u_\ell, L_0, k-h)$ for $1 \leq h \leq k$ where a set $x_{i,j}$ is constructed if there is a $y \epsilon \Sigma^*$ such that $f(q_0, y) = x_{i,j}$. The set $x_{i,j}$ will contain nothing else. Certainly this construction can be done since by the construction of S there must be a $g(u_\ell, L_0, k)$ as defined for each $p_i \epsilon P$. Furthermore, this construction partitions all of the sets of states in $A(S,k)$ since

no states can exist which are not of the form $g(u_{\ell}, L_0, k-h)$ for $0 \leq h \leq k$. This can be checked by studying the construction of S.

It will be necessary to use the following property which can be easily proved:

Property A. If $d(p_0, y_1) = d(p_0, y_2)$, then $g(y_1, L_0, k) = g(y_2, L_0, k)$.

Condition (1) of Lemma 3.2 will be proved by induction on the length of w.

I. If length $(w) = 0$ then $d(p_0, w) = p_0$ and $f(q_0, w) = \{q_0\} = \{g(\Lambda, L_0, k)\} \in \{x_{0,1}, \ldots, x_{0,j_0}\}$.

II. Assume $\text{length}(w) = h$, $\text{length}(wa) = h+1$, and condition (1) holds for $\text{length}(w) = h$. $d(p_0, wa) = p_i$ if and only if there is a p_j such that $d(p_0, w) = p_j$ and $d(p_j, a) = p_i$. This is true if and only if $f(q_0, w) \in \{x_{j,1}, x_{j,2}, \ldots, x_{j,j_j}\}$ and $d(p_j, a) = p_i$ by the induction hypothesis. It remains to be shown that the last stated conditions hold if and only if $f(q_0, wa) \in \{x_{i,1}, x_{i,2}, \ldots, x_{i,j_i}\}$.

That this is true can be seen by examining the states of $x \in \{x_{j,1}, \ldots, x_{j,j_j}\}$ and observing what happens as the input symbol a is applied. First of all we know that $g(u, L_0, k) \in x$ where $d(p_0, u) = p_j$

by the construction of x . Since the set $S_1 = u \cdot g(u,L_0,k+1)$ is a subset of S and $S_2 = ua \cdot g(ua,L_0,k)$ is a subset of S because $S_2 \subseteq S_1$, we have $g(ua,L_0,k) \in f(g(u,L_0,k),a)$ by definition of f . (One case which deserves special comment is when $g(ua,L_0,k) = \varphi$. This is dealt with in the next paragraph.) But $d(p_0,ua) = p_i$ so the newly found state $g(ua,L_0,k)$ is exactly the state which was incorporated into all of $x_{11}, x_{12}, \ldots, x_{1,j_1}$. This assertion uses Property A and the fact that $g(u',L_0,k)$ was included in every set $x_{11}, x_{12}, \ldots, x_{1,j_1}$ where $d(p_0,u') = p_i$. Similarly the states obtained by computing $f(q,a)$ for all other q in x can be examined and shown to be of the form $g(u',L_0,k-r-1)$ if $q = g(u,L_0,k-r)$. It is left to the reader to fill in the remaining details and thus verify that

$f(q_0,wa) \in \{x_{11}, x_{12}, \ldots, x_{1,j_1}\}$.

Referring to the previous paragraph, if $g(ua,L_0,k) = \varphi$ then one of the two cases will follow:

<u>Case I.</u> $g(ua,L_0,k+1) \neq \varphi$. If $g(ua,L_0,k) = \varphi$ and $uaw \notin S$ for any $w \in \Sigma^*$, then $g(ua,S,k)$ will be undefined and the proof will fail at this point. This is why it is necessary to have the term σ_u included in the construction of S . If $g(ua,L_0,k) = \varphi$ and $g(ua,L_0,k+1) \neq \varphi$ then $\sigma_u = 1$ and $u \cdot g(u,L_0,k+2) \subseteq S$. Then $ua \cdot g(ua,L_0,k+1)$ is a subset of S and is not empty. Therefore there will be a w such that $uaw \in S$ so that $g(ua,S,k)$ will be defined and the proof will go through.

<u>Case II</u>. $g(ua,L_0,k+1) = \varphi$. This occurs if p_i is an absorbing

state. This is true if and only if $g(ua,L_0,k)$ is not defined which

happens if and only if $f(g(u,L_0,k),a)$ is empty. So (1) of the Lemma

follows in this case as well.

Conditions (2) and (3) of the Lemma are clearly true. If $p_i \in D$

then $\Lambda \in g(u,L_0,k)$ where $d(p_0,u) = p_i$. Then $g(u,L_0,k) \in F$ so that

$x_{i,j} \cap F \neq \varphi$ for all $j = 1,2,3,\ldots,j_i$. If $p_i \notin D$ then

$\Lambda \notin g(u,L_0,b)$ for any nonnegative integer b so that $x_{i,j} \cap F = \varphi$

for all $j = 1,2,3,\ldots,j_i$. This completes the proof of the Theorem.

BIBLIOGRAPHY

[1] J. A. Brzozowski, "Derivatives of Regular Expressions," _Journal of the Association for Computing Machinery_, Vol. 11, No. 4, pp. 481-494, 1964.

[2] J. A. Feldman, "First Thoughts on Grammatical Inference," Stanford A. I. Memo No. 55, August 1967.

[3] J. A. Feldman, J. Gips, J. J. Horning, S. Reder, "Grammatical Complexity and Inference," Technical Report No. CS125, Computer Science Department, Stanford University, June 1969.

[4] A. Gill, _Introduction to the Theory of Finite-State Machines_, McGraw-Hill Book Company, Inc., New York 1962.

[5] S. Ginsburg, "Synthesis of Minimal-State Machines," _IRE Transactions on Electronic Computers_, EC8, pp. 441-449, 1959.

[6] S. Ginsburg, _An Introduction to Mathematical Machine Theory_, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1962.

[7] S. Ginsburg, _The Mathematical Theory of Context-Free Languages_, McGraw-Hill Book Company, Inc., New York, 1966.

[8] M. A. Harrison, _Introduction to Switching and Automata Theory_, McGraw-Hill Book Company, Inc., New York, 1965.

[9] D. A. Huffman, "The Synthesis of Sequential Switching Circuits," _Journal of Franklin Institute_, Vol. 257, No. 3, pp. 161-190, 1954; No. 4, pp. 275-303, 1954.

[10] G. H. Mealy, "A Method for Synthesizing Sequential Circuits," _Bell System Technical Journal_, Vol. 34, No. 5, pp. 1045-1079, 1955.

[11] G. Ott and N. Feinstein, "Design of Sequential Machines from Their Regular Expressions," _Journal of the Association for Computing Machinery_, Vol. 8, No. 4, pp. 585-600, 1961.

[12] J. Reynolds, "Grammatical Covering," TM-96, Argonne National Lab., June 1968.

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*	2a. REPORT SECURITY CLASSIFICATION
Stanford Artificial Intelligence Project Computer Science Department Stanford University	Unclassified
	2b. GROUP

3. REPORT TITLE

On The Synthesis of Finite-State Acceptors

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

5. AUTHOR(S) *(First name, middle initial, last name)*

Alan W. Biermann and Jerome A. Feldman

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
April, 1970	31	12

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
ARPA SD-183	AIM-114
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. DISTRIBUTION STATEMENT

Statement No. 1 - Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

Two algorithms are presented for solving the following problem: Given a finite-set S of strings of symbols, find a finite-state machine which will accept the strings of S and possibly some additional strings which "resemble" those of S. The approach used is to directly construct the states and transitions of the acceptor machine from the string information. The algorithms include a parameter which enable one to increase the exactness of the resulting machine's behavior as much as desired by increasing the number of states in the machine. The properties of the algorithms are presented and illustrated with a number of examples.

The paper gives a method for identifying a finite-state language from a randomly chosen finite subset of the language if the subset is large enough and if a bound is known on the number of states required to recognize the language. Finally, we discuss some of the uses of the algorithms and their relationship to the problem of grammatical inference.

DD FORM 1473 (1 NOV 65)